CHAPTER 4

EFFICIENT ACCESS OF DATA RESOURCES IN CLOUD
4.1 INTRODUCTION

Data storage is the most useful and important service being offered by cloud, where in the customers need not store their personal data on their desktops / servers instead the same is stored in cloud. The cloud users will be charged based on the usage of memory space on the cloud. Here, storage is provided as a service which is scalable and high flexible, where the customers pay only for the storage space what actually they are using for a particular period of time. In case of storage as a service the user need not worry on the storage maintenance issues and also the installation of file systems. But, the actually data may be stored in different place and the users will not be aware of exact location of data storage, but can access their data from any where at any time through cloud provider’s network facility or via Internet. The major concern here is the data integrity, privacy and security.

As data retrieval involved either reports from the database or some queries on the stored database, some common data access APIs are required. Typically, the data access will be via a intermediate layer called as Database Management System. The DBMS
packages will act as an interface between data users and the physical data storage. In case of cloud computing also, the same concept is applied, but here the user need not purchase the commercial software like Oracle, indeed the user can use Oracle on cloud and pay for the time the used actually used Oracle. Hence, the major hurdle of commercial software procurement turns into a rental model.

**Cloud Data Access Scheme:** Accessing the cloud is slightly tricky than a traditional desktop enabled software usage. Unix operating system provides a beautiful facility to improve desktop performance namely memory virtualization. It is actually an old operating system concept used for enhancing scalability. The same concept is extended with enhanced facilities and features provide hardware scalability in cloud computing. One such example is *Desktop-as-a-Service* that is suitable for WAN.

**Desktop as a Service:** While cloud provides *Desktop as a Service*, the users can access all the typical desktop computing and user interface services from remotely located servers. As traditional desktops are basically meant for user interface to utilize and manage the computing resources including both hardware and software, there are certain changes in the type of utility when it comes to remote desktops. In case of cloud computing, single machine / server is shared by many users to achieve better utilization, hence all the resources need to be properly mapped to the users and the user cannot enjoy the complete freedom of hardware as in case of traditional desktops. There are several challenges in this regard that include customer satisfaction, scalability, ensuring maximum resource utilization and negligible SLA violation. When many customers are assigned single server, it leads to dissatisfaction among customers, while if less are assigned leads under utilization of resources and higher investment cost. In desktop oriented services typically the task processing is sequential and leads to huge response time and hence down grades the whole system performance. Since traditional desktop interfaces use dedicated hardware and pre-installed software tools, the access to system utilities is denied to the user without such software. But, in case of virtual desktop the cloud computing removes this requirement and hence ease of use.

### 4.2 VIRTUALIZATION

Before the evolution of cloud, virtual networks are used connect multiple computers to perform high performance computing. Since it was easier and compact, the same concept
of virtual networks is extended and renamed as virtualization [48]. It allows multiple instances on system images on one server, i.e., one machine can serve as multiple servers or OS which distributes the task into several cloud based segments. Like this, a given task is separated into several sub tasks and finally, the results of all such sub tasks are accumulated to provide desired optimal solution.

**Role of Virtualization in Cloud Sector:** As computational tasks are becoming more and more time consuming and highly complex in nature, with advent of newer hardware technologies, fully functional communication systems, and the emergent of high performance computing platforms like cluster computing it is required to use distributed data and resources between various levels of abstraction. Virtualization technology enables to divide both hardware and software resources evenly via centralized data servers in cloud environment [61].

It is possible to access data by virtualization from multiple OS or servers instances which are actually being located on single physical server. It can also be treated like a single OS being served to multiple VMs to perform distributed jobs. Since virtualization hides the characteristics of physical resources from the cloud users, they can use these resources like a black box. Because, the users are more concerned to use the resources than that of knowing more details of those resources.

Virtualization use hypervisors which are nothing but virtual managers to control data flow through many incoming requests [115]. Actually, hypervisor is utility software which manages multiple instances of one operating system. Hypervisors are nothing but cloud operating systems which manage cloud resources, memory, CPU speed. The basic motive behind virtualization is reliability, easy accessibility and cheap cost to help IT industry.

**4.3 HYPERVISORS**

Technically hypervisors are the firmwares developed to manage VMs. Hypervisors are of two types namely *Type I* that runs on the base system and *Type II* is basically an emulator. The examples for *Type I* hypervisors are RTS hypervisor, LynxSecure, Oracle VM, Sun xVM Server, VLX, VirtualLogic etc. The figure 4.1 gives an overview of *Type
I hypervisor. There will not be any host OS for Type I hypervisors as they run on a bare system.

**Figure 4.1: Hypervisor Type 1**

Type II hypervisors are emulator interfaces or containers that interact with the hardware. The examples include Microsoft Hyper V, VMWare Workstation 6.0, VMWare Fusion, KVM Windows Virtual PC, etc. The figure 4.2 gives an overall understanding of Type II hypervisor.

**Figure 4.2: Hypervisor Type 2**

In many available literature, it is also commented that cloud computing is a derivative of virtualization technology. Virtualization is basically meant for manipulation of hardware, where as cloud computing is more than hardware virtualization, which provides significant support for business decisions. Moreover, virtualization is to deal with software technology, where as cloud is meant for business enhancement. Of course, cloud uses virtualization technology as one of its inbuilt feature to provide better
scalable and flexible services to its customers. The definitions of virtualization and cloud computing is as follows:

“Virtualization is a basic element of cloud computing and helps cloud service providers to deliver on the value of cloud computing, and actually cloud computing is to do more with the delivery of shared software, data and computing resources in an on-demand service delivery model through Internet”.

The confusion occurs mainly in private clouds, where in majority of the cloud job is done with virtualization technology, where as in public clouds there is more beyond virtualization. Therefore, the actual difference is that the basic cloud provides elasticity, automatic resource management, scalability, self-service capacity and charge per usage model of service over internet, where as the virtualization is meant only to deal with hardware scalability, elasticity and better resource utilization. Figure 4.3 illustrates the basic working principle of virtualization where in the single physical server is placed in the middle and being connected to different virtual machines to perform tasks faster.

![Figure 4.3: Illustration of Virtualization Process](image)

### 4.4 TYPE OF VIRTUALIZATION TECHNIQUES

Application virtualization, hardware virtualization and full virtualization are the main three types of virtualization. In case of memory virtualization, the physical memory will be mapped to individual process memory or to the actual machine memory when multiple VMs are running on the single system. It uses shadow paging technique and
helps the cloud users to recover their data from cloud on system failure. In OS level virtualization the instances of physical server are mapped to multiple isolated partitions, where in each partition looks like a single server. Here, the OS kernel runs a single OS and gives OS functionality to each of the partition. But, OS Virtualization facilitates the system software of the cloud applications to use cloud hardware resources to run multiple instances of different systems simultaneously, where in the systems are independent and isolated from each other [116].

There are three types of hardware virtualization namely para-virtualization, emulation virtualization and full virtualization. In case of full hardware virtualization, the base hardware is completely simulated and the guest software does not require any further modifications to run. Figure 4.4 briefs the functioning of full hardware virtualization technology.

![Figure 4.4: Full Virtualization Technique](image)

In para hardware virtualization technique, data is accessed from the cloud by modifying OS kernel as virtualized instructions. It involves changing the OS kernel to replace some of the non-virtualized instructions by making hyper calls that communicated with hypervisor. It is different from full hardware virtualization, where in the original OS is not aware whether it is a virtualized call or the OS call which are eventually trapped using binary translation. Figure 4.5 summarizes the functionality of para-virtualization technology and also illustrates the functioning of emulation virtualization.
Finally, in case of emulation virtualization the VMs simulate the underlying hardware and are independent of the same. Here, the guest OS do not require any further modification.

### 4.5 MEMORY VIRTUALIZATION

Virtualization, a big IT innovation is nothing but the abstraction of IT resources which separates its physical instances and boundaries from its function. After the server, desktop and storage virtualization techniques, IT revolution started even in the niche area of application virtualization. Virtualization techniques have changed the mode of software development and encouraging service oriented IT delivery models.

The major benefit of virtualization is obtained through consolidation, where in less number of servers are used to run multiple applications through better networked storage utilities which results in major cost savings. Therefore, virtualization plays a major role in changing the economics of IT. As memory is an integral part of any digital electronic gadget like appliances, server, switches, routers, etc., many times it becomes the bottleneck while achieving performance.
As memory is becoming very cheap and available in abundance and cache memory is being used to achieve better performance, computer systems are now available with higher memory capacities. But, there are several issues like cache coherence problem and concurrency are still the niche area or research. Especially, when multiple applications try to use the same physical memory during virtualization several problems arise. Initially, memory virtualization was introduced to decouple memory from the shared processor in a multi-tasking or distributed computing environment. Even though the virtual memory is not physically addressable to facilitate memory sharing among multiple machines, the major issue arises with synchronization. Actually, memory and storage are not synonymous and memory virtualization is more focused on better application performance. It is a fact that the CPU directly access data from the memory and the storage is static and persistent regardless of the type of physical memory. Memory virtualization is used extensively in IT applications to extend the main memory beyond its actual capacity, to implement shared memory in a distributed computing environment, and to enable cloud and real time IT infrastructure in the data centres. Figure 4.6 depicts the
proposed cloud architecture that employs *Desktop-as-a-Service* for data access from cloud by using para-virtualization.

### 4.6 FEATURES OF THE PROPOSED ARCHITECTURE

- Data access from single machine and available for multiple remote machines through process of para-virtualization. It is operating system assisted virtualization.
- Connected servers in data center to enable virtualized environment.
- The benefits of memory virtualization are passed on to application virtualizations service which confirms the performance related issues like throughput, reliability, durability as indicated in SLAs.
- Monitoring through Service Manager to ensure the flow of resources as required by the applications and thus monitoring the client request.
- The requests from applications are dealt by the operating systems and then OS is responsible for mapping these requests to hardware requirements.
- For making them adaptable to operating system requirements, para-virtualized OS is used. It modifies OS kernel and replaces non-compatible requests with hypercalls that communicates with hypervisor.
- As hypervisor is virtualization manager on physical hardware, so it fulfills multiple client requests for various resources and runs multiple applications.
- These resources are stored in form of various applications and are provided to multiple remote devices like laptops, mobile phones etc.
- SaaS allows multiple clients to access the cloud resources.

### 4.7 SHADOW PAGE TABLE TO EASE VIRTUALIZATION

Generally, shadow page table is used to maintain the mapping of physical address to guest virtual address. They are used by the hypervisors to track the state in which the guest thinks its page tables should be. The guest can't be allowed access to the hardware page tables because then it would essentially have control of the machine. So, the hypervisor keeps the real mappings (guest virtual→host physical) in the hardware when the relevant guest is executing, and keeps a representation of the page tables. Guest OS will maintain its own virtual memory page table in the guest physical memory frames. For each guest physical memory frame, VMM should map it to host physical memory frame. If a fault
occurs in a system due to system failure, the recovery manager utility of memory virtualization is invoked for possible recovery of lost data. Shadow paging is also used as one of the recovery method that works implicitly in accordance with memory status.

Shadow paging divides the virtual database into pages after mapping the instructions from virtual to physical machine and the mapping with physical address is maintained in the dedicated data structure called as *Shadow Page Map Table*. As and when the update operation is done, the data is copied into a new physical location and same is maintained in shadow page map table as illustrated in figure 4.7.

![Shadow Page Map Table](image)

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Old Address</th>
<th>New Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>103</td>
<td>500</td>
</tr>
<tr>
<td>Page 2</td>
<td>105</td>
<td>498</td>
</tr>
<tr>
<td>Page 3</td>
<td>108</td>
<td>504</td>
</tr>
</tbody>
</table>

**Figure 4.7: Shadow Paging Technique**

Analysis of results using shadow paging virtualization technique is shown in the graph as in figure 4.8. The major issues with shadow page table are given below:

- Caching guest page tables takes memory.
- Need to determine when guest has reused page tables.
• Guest may not need invalidate TLB on writes to off-line page tables
• Need to trace writes to shadow page tables to invalidate entries

![Evaluation of new and old memory addresses](image)

**Figure 4.8:** Analysis of Results using Shadow Paging Virtualization Technique

To appreciate the benefits of virtualization, the major difference between the private and public cloud need to be understood. Private cloud is owned by individuals who own their software and hardware basically used for their consumption, where as in public clouds, the resources are owned by third party and the consumers will get the cloud service based on pay as you use model. Therefore, in public clouds, many users are allowed to use their resources in a multi-tenant model.

The virtualization can be better exploited in case of private clouds as the organization has owns its resources and hence the user gets the best control and flexibility to manage their compute systems, while still providing cloud consumption features. In contrast, as public cloud is being rented to multiple users, there are some risks like having ugly neighbors and performance latency issues associated with virtualization on public clouds. In spite of several security and privacy threats with virtualization techniques, there are several advantages when the same is used in conjunction with cloud computing paradigm.
• **Maximum Utilization of Resources** – With virtualization techniques, under utilization of physical resources can be avoided and hence gives better value addition to the servers and also brings down the hardware investment.

• **Use of Multiple Systems** – Multiple applications are allowed to run on different Operating Systems on the same physical hardware.

• **Integration of IT Budget** – As virtualization brings down the hardware requirements, the corresponding administration, management and dependent requirements of managing IT infrastructure come down drastically.

### 4.8 ISSUES AND CHALLENGES IN VIRTUALIZATION

• **Backup and Recovery in Virtualization** – The data recovery from the virtualized servers is slightly different than in the case of traditional standalone servers. Here, frequent backups may eventually end up in data loss. Some commercial vendors like VDDK (Virtual Disk Development Kit) provide backup and recovery solutions to cloud customers. But, they are expensive and their security is still under question.

• **VM Sprawl** – Virtualization may sometimes lead to the out-of-control situation and some pop-up blockers allow advertisements. It may hamper the performance of cloud applications.

• **Virtual Capacity Planning** - To predict the futuristic behavior of cloud resources, budget and acquisition strategies, the hardware resources like CPU, Memory, Storage, and I/O are to be managed efficiently. But, it is very difficult in case of virtualization.

• **VM Stall** – As more and more organizations are deploying virtual machines in large numbers, VM stall may occur due to uncontrollable VM management, performance degradation and lack of trust in cloud administration.

• **Building Private Cloud** – As deploying VMs to private cloud is a rigorous task because of requirement of heavy infrastructure, geographical set-up and ill defined strategies for building private cloud, it is not feasible solution.
4.9 STRING MATCHING ALGORITHMS

This section attempts to give a new string matching designed based on a two dimensional structure on a cloud atmosphere. In any string matching algorithm, the algorithm takes one or more string patterns as input, then will execute on a large string or text and finally gives the binary output along with the position of the string.

Data retrieval basically deals with report generation and answers to queries from an underlying database. Retrieval engine retrieves the data from the database for a given user query. The DBMS software usually manages the backend database, select the subset of the data which answer the query, which can be stored in the form a file or viewed on computer screen. Retrieving relevant data from the back end database is very much used for attribute editing, updating, querying, data analysis and finally display. Data storage and retrieval subsystem involves user defined querying and data retrieval as one of its integral part.

String searching plays a vital role in many computational problems that include symbol manipulation, text editing and data retrieval. Inspite of indexing techniques being used to search huge amount of text, string searching algorithms help in information retrieval systems. String search algorithms are used for many filtering techniques to reduce the search space and also for searching some highlighted outputs. String matching techniques are used in many advanced text processing and also used in some system software like editors which are the basic of many operating system. In text processing the user request to search for a pattern is searched again the entire text. For example, in Microsoft Word Processor or even in an HTML page on a Mozilla web browser you can search for a string in the entire documents using search (Ctrl+F) option. In Unix grep command is used for basis text search.

The input pattern is given as \( x = x[0…m-1] \) with length of the string \( x \) denoted as \( m \) and the repository string on which search has to occur is represented as \( y[0…n-1] \) with its length denoted as \( n \). Both \( x \) and \( y \) are the subsets of the language build over the alphabet \( \Sigma \).

String matching either includes single pattern matching or multiple pattern matching. Few techniques are based on the concept of approximate pattern matching wherein the text is matched to the pattern to an approximate i.e., we say that a match has occurred
when an approximate similarity of the pattern with the text occurs. Depending upon the
number of pattern each algorithm uses, the algorithms are classified. Simple algorithms
like Naïve or Brute-force techniques are of linear time complexity and hence more time
consuming. Extension of number-theoretic notations that include modulus operations
on numbers are used in Rabin-Karp string matching algorithm. Disadvantage of another
popular string matching algorithm called Knuth-Morris-Pratt algorithm is that of linear
time complexity [117], wherein it uses a prefix function π to encapsulate knowledge on
window based pattern matching. Currently, the Boyre-Moore algorithm and its slight
variants are used in some applications because of its sub-linear searching time. Boyre-
Moore algorithm takes two functions, that is one is a bad character and another is a
good prefix function that require some pre-processing. Here, we consider m as the
length of a sentence and n as the length of the file to be searched for.

Some of the examples of single pattern algorithms are (i) Naïve based algorithm, (ii)
Rabin Karp algorithm, (iii) Boyer-Moore algorithm, and (iv) Bitmap algorithm.
Similarly, the algorithms that use finite set of patterns are (i) Aho-Corasick algorithm,
(ii) Commentz-Walter algorithm and (iii) Rabin-Karp algorithm.

Knuth-Morris-Pratt algorithm was developed in 1970 and the first among the string
matching algorithms that has both best and worst time complexity as linear irrespective
of the length of the pattern [38]. The pre-processing time complexity of the algorithm is
$O(m)$ and the estimated number of comparisons performed by the algorithm is time
bounded by the following equation.

$$ n + O(1) \approx \bar{C} \leq 2n + O(1) $$

The basis of this algorithm is when a mismatch occurs; the false start consisting of the
characters that are already used for comparison will take place and repeated over every
mismatch, which is in-turn advantageous against the run time complexity. Again, the
algorithm can re-arrange the pointer, so that the pointer is not disturbed. To make this
possible, the pattern that is already pre-processed to get the table required, gives the
next position in the same pattern to be processed after a mismatch. The pseudocode of
the algorithm is given below:

$$ \text{next}_j = \max \{ i \mid \text{pattern}_k = \text{pattern}_{j+i+k} \text{ for } k = 1 \ldots i-1 \} $$
and \( \text{pattern}_i \neq \text{pattern}_j \) for \( j = 1 \ldots m \). To put it in other way, the maximal matching prefix of the pattern is used in such a way that the next character in the pattern is different from the character of the pattern that made the mismatch. To illustrate, the following table is constructed for the pattern abracadabra.

\[
\begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{r} & \text{a} & \text{c} & \text{a} & \text{d} & \text{a} & \text{b} & \text{r} & \text{a} & \\
\text{next}[j] & 0 & 1 & 1 & 0 & 2 & 0 & 2 & 0 & 1 & 1 & 0 & 5 \\
\end{array}
\]

When the value in the \( \text{next}[j] \) table becomes 0, the position in the text is advanced by one position and starts again the comparison from the starting position of the pattern. The final value of the \( \text{next} \) table is used to restart the search after the match has found.

Boyer-Morre string matching algorithm was developed in 1977, where the search happens from right to left, and the time complexity is better than average. It positions the pattern on the leftmost character in the given text and tries to find the match from right to left \([117]\). If mismatch does not occur then the pattern is found, otherwise, the algorithm computes a shift of the number of characters towards right on the input pattern, prior to undertaken a new matching attempt, the shift is performed by match heuristics and the occurrence heuristic. When the pattern is moved to the right, the match heuristic is obtained and it must be matched to all the characters that are previously matched and should bring a character to the position in the text that caused the mismatch. This condition was introduced to Boyer-Moore algorithm by Knuth in 1977. The formal definitions are given below:

\[
\hat{d}_d[j] = \min\{s + m - j | s \geq 1 \text{ and } ((s \geq i \text{ or } \text{pattern}[i - s] = \text{pattern}[i]) \text{ for } j < i \leq m}\},
\]

for \( j = 1, \ldots, m \); and

\[
\hat{d}_d[j] = \min\{s + m - j | s \geq 1 \text{ and } (s \geq j \text{ or } \text{pattern}[j - s] \neq \text{pattern}[j]) \text{ and } ((s \geq i \text{ or } \text{pattern}[i - s] = \text{pattern}[i]) \text{ for } j < i \leq m}\}.
\]

The following is the example table constructed on the pattern \( \text{abracadabrais} \)

\[
\begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{r} & \text{a} & \text{c} & \text{a} & \text{d} & \text{a} & \text{b} & \text{r} & \text{a} & \\
\hat{d}_d, & 17 & 16 & 15 & 14 & 13 & 12 & 11 & 13 & 12 & 4 & 1 \\
\end{array}
\]
Next, in case of occurrence heuristic, first align the position of the pattern that caused the mismatch with the first one that caused the match. The pseudocode for the same is given below.

\[ d_x = \min\{s | s = m (0 \leq s < m \text{ and the pattern } [m-s] = x) \} \text{ for each symbol } x \text{ in the alphabet.} \]

The main features of the Karp-Rabin algorithm is that it (i) uses the hashing function, (ii) takes \(O(m)\) time to pre-process, (iii) takes \(O(mn)\) time to search, (iv) finally takes \(O(n+m)\) expected running time. As hashing is a simple technique to avoid quadratic time complexity to compare characters, instead of comparing every position of the given text against the occurrence of a given key pattern, it seems to be better than checking in the hash bins. Hashing is more often used to check the resemblance among the words. To use hashing functions in string matching algorithms following properties are to be considered, i.e., efficient computability, discrimination for strings, \(\text{hash}(y[j+1 \ldots j+m])\) are to be easily derived from \(\text{hash}(y[j \ldots j+m-1])\) and \(y[j+m]\), and finally \(\text{hash}(y[j+1 \ldots j+m])\) are to be rehashable from the inverse function \(\text{rehash}(y[j], y[j+m], \text{hash}(y[j \ldots j+m-1]))\). Here, the length of the string \(w\) of length \(w\) is taken as \(\text{hash}(w)\) and defined as follows:

\[
\text{hash}(w[0 \ldots m-1]) = (w[0]*2^{m-1} + w[1]*2^{m-2} + \ldots + w[m-1]*2^0) \mod q, \text{ where } q \text{ is a large number. Then, finally } \text{rehash}(a, b, h) = ((h-a*2^{m-1})*2+b) \mod q
\]

In the pre-processing step of Karp-Rabin algorithm, we compute \(\text{hash}(x)\) and can be achieved in constant space and \(O(m)\) time. During this search phase, comparing \(\text{hash}(x)\) with \(\text{hash}(y[j \ldots j+m-1])\) for \(0 \leq j < n-m\) is enough and if found equal, the check for equality \(x=y[j \ldots j+m-1]\) character by character.

DFA is used to search a given word \(x\), where minimal DFA denoted as \(A(x)\) is constructed by recognizing the particular language \(\Sigma^*x\). \(A(x)\) is denoted as a quadruple \((Q, q_0, T, E)\) that recognizes the language covered by \(x\) of \(\Sigma^*x\), where \(Q\) is a prefix set of \(x\) denoted as \([\varepsilon, x[0], x[0 .. 1] \ldots x[0 .. m-2], x]\), \(q_0=\varepsilon\) and \(T=\{x\}\). For a given state \(q\) over the set of all prefixes i.e., \(Q\) and \(a\) in the set of alphabets \(\Sigma\), the the triplet \((q, a, q_a)\) belongs to \(E\) iff \(q_a\) is a prefix of \(x\). If not the triplet \((q, a, p)\) belongs to \(E\) in such a way that \(p\) is the longest suffix of \(q_a\) which is a prefix of \(x\). The DFA \(A(x)\) is built with the time complexity of \(O(m+\sigma)\) and space complexity \(O(m\sigma)\).
Once $A(x)$ is constructed, searching for a given word $x$ in a text repository $y$ involves parsing the entire text $y$ with $A(x)$ with the initial state $q_0$. An occurrence of $x$ is reported whenever the terminal state is encountered. The time complexity of the searching phase is $O(n)$ and that of automaton that is stored in direct access table is $O(n \log(\sigma))$. The analysis of the asymptotic time complexity of different algorithms is given in Table 4.1.

<table>
<thead>
<tr>
<th>String Matching Algorithm</th>
<th>Time complexity for Pre-processing</th>
<th>Asymptotic Notation of the Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve based string matching</td>
<td>Zero as it does not require pre-processing</td>
<td>$\Theta((n-m+1) m)$</td>
</tr>
<tr>
<td>Rabin-Karp algorithm.</td>
<td>$\Theta (m)$</td>
<td>Average Case $\Theta(n+m)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Worst Case $\Theta((n-m+1) m)$</td>
</tr>
<tr>
<td>DFA based search</td>
<td>$\Theta (m*</td>
<td>\Sigma</td>
</tr>
<tr>
<td>Knuth-Morris-Pratt algorithm</td>
<td>$\Theta (m)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Boyre-Moore algorithm</td>
<td>$\Theta (m +</td>
<td>\Sigma</td>
</tr>
<tr>
<td>Bitmap algorithm that uses multiple shifts</td>
<td>$\Theta (m +</td>
<td>\Sigma</td>
</tr>
</tbody>
</table>

A logical matrix is a binary matrix that can represent a relation or a Boolean expression from the domain $\{0, 1\}$, which is basically used to represent a binary relation over finite sets. If a given relation $R$ is binary on the finite sets $X$ and $Y$, then the relation $R$ can be represented as a matrix $M$, where the rows depict $X$ and columns depict $Y$ and the entries of $M$ are defined as follows:

$$M_{i,j} = \begin{cases} 1 & (i,j) \in R \\ 0 & (i,j) \not\in R \end{cases}$$

### 4.10 PROPOSED TECHNIQUE

Matrix calculus is used in the proposed technique, where the grid is drawn across the rows and the columns. Here, the input text and the pattern entered by the user is set to a grid pattern depending on the length of the text. Both the text and pattern gets converted into a matrix from wherein the text appears across the column and the pattern across the row. Once the text and the pattern have been set, then the matching starts.
The entries in the matrix are either a “1” or “0” depending upon the match. If there exists a match between the character of the given text and that of a pattern then the value inserted in the matrix is 1 else it is set to 0. Suppose, let the text be a series of numbers like 9 1 2 7 3 4 5 6 3 1 2 8 8 3 1 2 and let the pattern be 3 1 2 then it works as follows using a simple logical matrix. The working procedure is illustrated in figure 4.9.

Text = 9 1 2 7 3 4 5 6 3 1 2 8 8 3 1 2

Pattern= 3 1 2

j = Position of characters in text

i = Position of characters in pattern

\[
\begin{array}{cccccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
0 & 3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
2 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

Figure 4.9: First Approach to Find Pattern

Problems with First Approach: In the above mentioned technique it is clear that whenever a match is encountered a 1 is inserted else a 0. According to this technique, a pattern exists in a text if there is a diagonal consisting of all 1’s and the length of the diagonal is equal to the number of characters in pattern.

But this technique is not very efficient because there exists problem of spurious hits. It is evident that there are many 1’s in the matrix but they don’t form the pattern. This problem gets amplified and time consuming when one will like to find the position of the pattern. The position of the pattern will be found from the last row. In the last row it can be seen that there exists three 1’s but only two 1’s will lead to correct pattern, the
other 1 will be a spurious hit because if backtracking will be done then at the last it will not be the pattern one is searching for.

The above mentioned technique can be useful if one wants to do approximate string matching. The approximate string matching technique is the one wherein the technique searches for approximate patterns and not exact patterns found in the text. Like in the above example it is clear that 312 is the exact pattern and 112 is an approximate pattern in the text.

**Second Approach:** The refined technique (*MatMatcher*) is an exact pattern matching technique and works in the same way as above but with little changes. In this, the entries are inserted by referring the previous row i.e. whenever a 1 is to be entered then the previous row’s upper diagonal element is checked and if it is 1 then only a 1 is inserted else a 0 is entered. The advantage of using this approach is that, there are no spurious hits. Also for finding the number of occurrences one needs to refer to the last row and not the entire matrix.

A new technique called *MatMatcher* has been successfully developed. *MatMatcher* is an efficient technique used for searching strings that matches or searches for a particular pattern entered by the user. The algorithm for the above mentioned technique has been designed, which can be easily implemented in any programming language ranging from C to Java. The technique is capable of matching patterns of numbers or alphabets both. *MatMatcher*, effectively uses the logical matrix to insert the values that indicate the presence and absence of pattern in the text. Only Boolean values have been used i.e. 1 and 0. Two techniques have been discussed for string matching in which the second one is more efficient than the first as the first technique suffers from the problem of spurious hits. The first technique can still be used for approximate string matching. The second technique is an exact string matching technique.

In order to find the position of pattern in the given text (as illustrated in figure 4.10), the following formula is used.
\( n = \text{Length of the Text} = 17; \ m = \text{Length of the Pattern} = 3; \ \text{Position of pattern in text} = j = m + 1; \) For \( j=11 \) The Position = 11-3+1 and the hence Pattern appears at position 9 in Text.

For \( j = 16; \) The Position = 16-3+1 = 14; hence Pattern appears at position 14 in Text.

Figure 4.10: Second Approach to Find Pattern

4.11 ALGORITHM MatMatcher

MatMatcher\( (P, T, n, m) \)

\( n \) – length of text; \( m \)- length of pattern; \( P \)- pattern; \( T \)- text;

\( a[ ][] \) – array

size of array – \( m \ast n \)

Step 1: Read the text

Step 2: Read the pattern

/*obtaining values in terms of 0s and 1s for the first row of matrix */

Step 3: for\( (j=0; j<n; j++) \) {
    if \( (p[0] == t[j]) \) {
        \( a[0][j] = 1; \)
    }
    else {
        \( a[0][j] = 0; \)
    }
}
/*obtaining the values of the remaining rows */

**Step 4:** for(i=1; i<m; i++) {
    for(j=0; j<n; j++) {
        if (p[i] == t[j] & (a[i-1][j-1]) == 1)) {
            a[i][j]=1;
        }
        else {
            a[i][j]=0
        }
    }
}

/*extracting the last row for checking the matching of pattern with text*/

**Step 5:** k = 0;

**Step 6:** for (j=0; j<n; j++) {
    if(a[m-1][j] == 1) {
        Print “Pattern appears in Text at position: j-m+1”
        k=k+1;
    }
}

**Step 7:** if(k = 0) Print “Pattern not found”
else Print “Pattern appears k times in text”

**Step 8:** End.

**Performance Analysis:** There are in all three loops out of which two loops are nested. Out of the two nested loops the outer loop is running from 1 to m-1 and second from 0 to n-1 and hence the complexity turns out to be O((m-1)n). Similarly, the space complexity consists of the matrix which is a structure composed of rows and columns.
The proposed technique is using a matrix structure and hence it is composed of $m$ rows and $n$ columns, wherein $m$ is the number of characters in the pattern and $n$ is the number of characters in the text. So accordingly the complexity comes out to be $m \times n$. In general terms we may conclude that the space complexity is $n^2$. The response time of the proposed algorithm against the varying length of patterns is as shown in figure 4.11.

![Figure 4.11: Average Response Time versus Length of the Pattern](image)

**SUMMARY**

A new technique called *MatMatcher* has been successfully developed. *Matmatcher* is an efficient technique used for searching strings that matches or searches for a particular pattern entered by the user. The algorithm for the above mentioned technique has been designed, which can be easily implemented in any programming language ranging from C to Java. The technique is capable of matching patterns of numbers or alphabets both. *MatMatcher*, effectively uses the logical matrix to insert the values that indicate the presence and absence of pattern in the text. Only Boolean values have been used i.e., 1 and 0. Two techniques have been discussed for string matching in which the second one is more efficient than the first as the first technique suffers from the problem of spurious hits. The first technique can still be used for approximate string matching. The second technique is an exact string matching technique.